UM1779
User manual

Getting started with STM32CubeF0 firmware package for STM32F0 series

Introduction

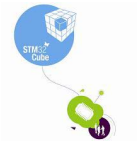
The STMCube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF0 for STM32F0 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, STMTouch, FatFS and Graphics
  - All embedded software utilities coming with a full set of examples.

This user manual describes how to get started with the STM32CubeF0 firmware package.

Section 1 describes the main features of STM32CubeF0 firmware which is part of the STMCube™ initiative. *Section 2* and *Section 3* provide an overview of the STM32CubeF0 architecture and firmware package structure.



Contents

List of tables

List of figures

1 STM32CubeF0 main features

STM32CubeF0 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32F0 microcontrollers. In line with the STMCube™ initiative, this set of components is highly portable, not only within STM32F0 series but also to other STM32 series.

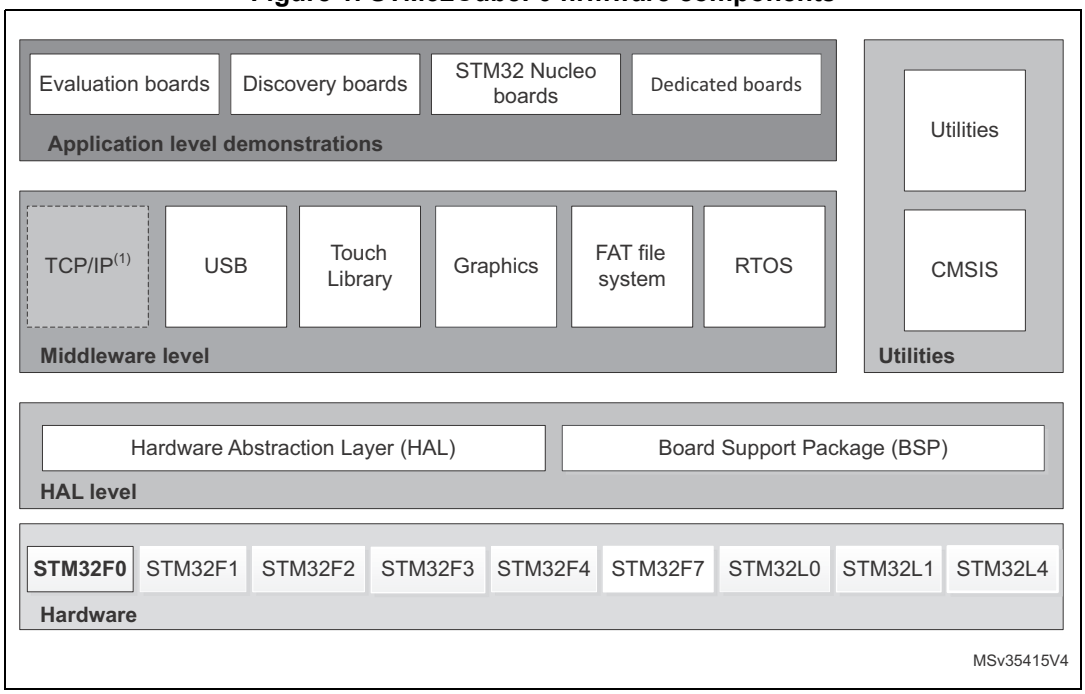
STM32CubeF0 is fully compatible with STM32CubeMX code generator that allows to generate initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for user convenience.

STM32CubeF0 package also contains a set of middleware components with the corresponding examples. They come in very permissive license terms:

- Full USB Device stack supporting many classes: Audio, HID, MSC, CDC and DFU.
- STemWin, a professional graphical stack solution available in binary format and based on STMicroelectronics partner solution SEGGER emWin.
- FAT File system based on open source FatFS solution
- STMTouch touch sensing library solution
- CMSIS-RTOS implementation with FreeRTOS open source solution.

Several applications and demonstrations implementing all these middleware components are also provided in the STM32CubeF0 package.

Figure 1. STM32CubeF0 firmware components

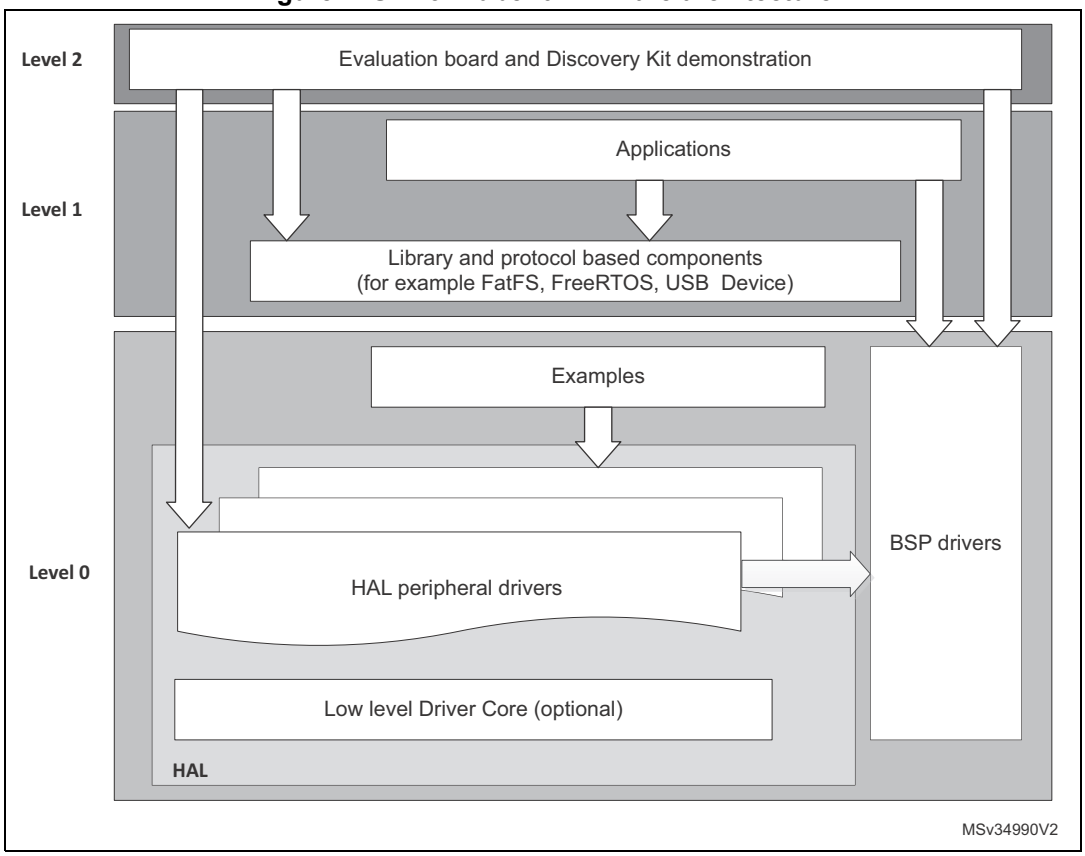


1. Not available in STM32CubeF0 package.

2 STM32CubeF0 architecture overview

The STM32Cube firmware solution is built around three independent levels that can easily interact as described in *Figure 2*.

Figure 2. STM32CubeF0 firmware architecture



Level 0

This level is divided into three sub-layers:

- **Board Support Package (BSP)**

  This layer offers a set of APIs relative to the hardware components in the hardware boards (LCD drivers, microSD, etc…). It is composed of two parts:

  – **Component**: this is the driver relative to the external device on the board and not related to the STM32. The component driver provides specific APIs to the BSP driver external components and can be ported on any other board.

  – **BSP driver**: it permits to link the component driver to a specific board and provides a set of user-friendly APIs. The APIs naming rule is BSP_FUNCT_Action():

  Example: BSP_LED_Init(), BSP_LED_On()

  It is based on a modular architecture allowing an easy porting on any hardware by just implementing the low level routines.

- **Hardware Abstraction Layer (HAL)**

  This layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic multi-instance feature-oriented APIs which simplify user application implementation by providing ready-to-use process. As example, for the communication peripherals (I2S, UART…) it includes APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and handle communication errors that may raise during communication. The HAL drivers APIs are split in two categories:

  – Generic APIs which provides common and generic functions to all the STM32 series

  – Extension APIs which provides specific and customized functions for a specific family or a specific part number.

- **Basic peripheral usage examples**

  This layer encloses the examples build over the STM32 peripheral using only the HAL and BSP resources.

Level 1

This level is divided into two sub-layers:

- **Middleware components**

  Set of Libraries covering USB Device library, StemWin, STMTouch touch sensing library, FreeRTOS, FatFS. Horizontal interactions between the components of this layer is performed directly by calling the feature APIs while the vertical interaction with the low level drivers is done through specific callbacks and static macros implemented in

the library system call interface. As example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.

The main features of each middleware component are as follows:

USB Device Library

- Several USB classes supported (Mass-Storage, HID, CDC, DFU)
- Support of multi-packet transfer features that allows sending big amounts of data without splitting them into maximum packet size transfers.
- Use of configuration files to change the core and the library configuration without changing the library code (Read Only).
- RTOS and Standalone operation
- Link with low level driver through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level drivers.

STemWin Graphical stack

- Professional grade solution for GUI development based on SEGGER's emWin solution
- Optimized display drivers
- Software tools for code generation and bitmap editing (STemWin Builder…)

FreeRTOS

- Open source standard
- CMSIS compatibility layer
- Tickless operation during low-power mode
- Integration with all STM32Cube Middleware modules

FAT File system

- FATFS FAT open source library
- Long file name support
- Dynamic multi-drive support
- RTOS and standalone operation
- Examples with microSD.

STM32 Touch sensing library

Robust STMTouch capacitive touch sensing solution supporting proximity, touchkey, linear and rotary touch sensor using a proven surface charge transfer acquisition principle.

- **Examples based on the middleware components**

  Each middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several Middleware components are provided as well.

Level 2

This level is composed of a single layer which consist in a global real-time graphical demonstration based on the middleware service layer, the low level abstraction layer and the basic peripheral usage applications for board based features.

3 STM32CubeF0 firmware package overview

3.1 Supported STM32F0 devices and hardware

STM32Cube offers highly portable Hardware Abstraction Layer (HAL) built around a generic architecture. It allows the build-upon layers, such as the middleware layer, to implement its functions without knowing, in-depth, the MCU used. This improves the library code re-usability and guarantees an easy portability on other devices.

In addition, thanks to its layered architecture, the STM32CubeF0 package offers full support of all STM32F0 series. The user only has to define the right macro in stm32f0xx.h.

Table 1 shows the macro to be defined depending on the STM32F0 device used. This macro must also be defined in the compiler preprocessor.

Table 1. Macros for STM32F0 series

Macro defined in stm32f0xx.h	STM32F0 devices
STM32F030x6	STM32F030F4, STM32F030C6, STM32F030K6
STM32F030x8	STM32F030C8, STM32F030R8
STM32F030xC	STM32F030CC, STM32F030RC
STM32F031x6	STM32F031C4, STM32F031F4, STM32F031G4, STM32F031K4, STM32F031C6, STM32F031F6, STM32F031G6, STM32F031K6
STM32F051x8	STM32F051K4, STM32F051C4, STM32F051R4, STM32F051K6, STM32F051C6, STM32F051R6,STM32F051K8, STM32F051C8, STM32F051R8
STM32F070x6	STM32F070F6, STM32F070C6
STM32F070xB	STM32F070RB, STM32F070CB
STM32F071xB	STM32F071V8, STM32F071CB, STM32F071RB, STM32F071VB
STM32F042x6	STM32F042F4, STM32F042G4, STM32F042K4, STM32F042T4, STM32F042C4, STM32F042F6, STM32F042G6, STM32F042K6, STM32F042T6, STM32F042C6
STM32F072xB	STM32F072C8, STM32F072R8, STM32F072V8, STM32F072CB, STM32F072RB, STM32F072VB
STM32F038xx	STM32F038C6, STM32F038F6, STM32F038G6, STM32F038K6
STM32F048xx	STM32F048C6, STM32F048G6, STM32F048T6
STM32F058xx	STM32F058K8, STM32F058C8, STM32F058R8
STM32F078xx	STM32F078CB, STM32F078RB, STM32F078VB
STM32F091xx	STM32F091CB, STM32F091RB, STM32F091VB, STM32F091CC, STM32F091RC, STM32F091VC
STM32F098xx	STM32F098CC, STM32F098RC, STM32F098VC

STM32CubeF0 features a rich set of examples and applications at all levels making it easy to understand and use any HAL driver and/or middleware components. These examples run

on the STMicroelectronics boards listed in *Table 2*.

Table 2. Boards for STM32F0 series

Board	STM32F0 supported devices
STM32072B-EVAL	STM32F072xB
STM32091C-EVAL	STM32F091xC
STM32F072B-DISCOVERY	STM32F072xB
STM32F0308-DISCOVERY	STM32F030x8
NUCLEO-F070RB	STM32F070xB
NUCLEO-F072RB	STM32F072xB
NUCLEO-F030R8	STM32F030x8
NUCLEO-F031K6	STM32F031K6
NUCLEO-F042K6	STM32F042K6
NUCLEO-F091RC	STM32F091xC

STM32CubeF0 support both Nucleo-32 and Nucleo-64 boards.

- Nucleo-64 boards support Adafruit LCD display Arduino™ UNO shields which embed a microSD connector and a joystick in addition to the LCD.
- Nucleo-32 boards support Gravitech 7-segment display Arduino™ NANO shields which allow displaying up to four-digit numbers and characters.

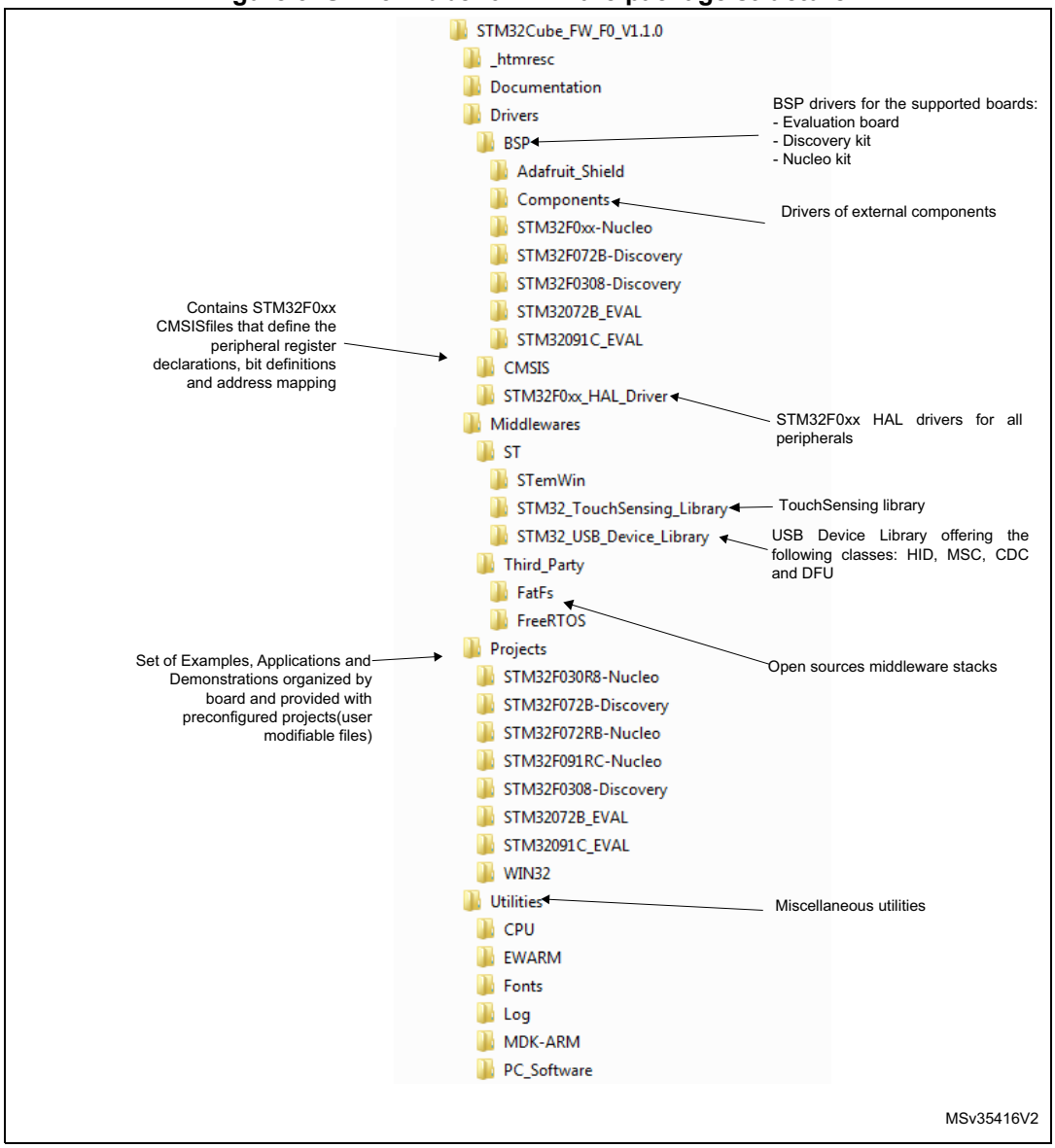
The Arduino™ shield drivers are provided within the BSP component. Their usage is illustrated by a demonstration firmware.

The STM32CubeF0 firmware can run on any compatible hardware. Simply update the BSP drivers to port the provided examples on your board if its hardware features are the same (e.g. LED, LCD display, buttons).

3.2 Firmware package overview

The STM32CubeF0 firmware solution is provided in one single zip package having the structure shown in *Figure 3*.

Figure 3. STM32CubeF0 firmware package structure



1. The components files must not be modified by the user. Only the \Projects sources can be changed by the user.

For each board, a set of examples are provided with pre-configured projects for EWARM, MDK-ARM and TrueSTUDIO toolchains.

Figure 4 shows the project structure for the STM32091C-EVAL board.

Figure 4. STM32CubeF0 examples overview

The examples are classified depending on the STM32Cube level they apply to, and are named as below:

- Examples in level 0 are called *Examples.* They use the HAL drivers without any middleware component.
- Examples in level 1 are called *Applications*.They provide typical use cases for each middleware component and can call several HAL drivers.

The *Template* project available in the Template directory allows to quickly build any firmware application on a given board.

All examples have the same structure:

- *\Inc* folder that contains all header files
- *\Src* folder for the sources code
- *\EWARM*, *\MDK-ARM* and *\TrueSTUDIO* folders that contain the pre-configured project for each toolchain
- *readme.txt* describing the example behavior and needed environment to make it working.

Table 3 gives the number of projects available for each board.

Table 3. Number of examples available for each board

Board	Examples	Applications	Demonstration
STM32072B-EVAL	40	16	N/A
STM32091C-EVAL	42	10	1
STM32F072B-DISCOVERY	43	5	1
STM32F0308-DISCOVERY	28	1	1
NUCLEO-F070RB	30	1	1
NUCLEO-F072RB	41	1	1
NUCLEO-F030R8	29	1	1
NUCLEO-F031K6	1	N/A	1
NUCLEO-F042K6	1	N/A	1
NUCLEO-F091RC	42	2	1

4 Getting started with STM32CubeF0

4.1 Running your first example

This section explains how simple it is to run a first example within STM32CubeF0. It uses as illustration the generation of a simple LED toggle running on STM32F091RC Nucleo board:

1. Download the STM32CubeF0 firmware package. Unzip it into a directory of your choice. Make sure not to modify the package structure shown in *Figure 3*. Note that it is also recommended to copy the package at a location close to your root (e.g. C:\STM32 or G:\Tests) because some IDEs encounter problems when path length is too long.
2. Browse to \Projects\STM32F091RC-Nucleo\Examples.
3. Open \GPIO, then \GPIO_EXTI folder.
4. Open the project with your preferred toolchain. A quick overview on how to open, build and run an example with the supported toolchains is given below.
5. Rebuild all files and load your image into target memory.
6. Run the example: each time you press the USER push button, the LED2 toggles (for more details, refer to the example readme file).

To open, build and run an example with the supported toolchains, follow the steps below:

- EWARM
  a) Under the example folder, open \EWARM sub-folder.
  b) Launch the Project.eww workspace[(a)].
  c) Rebuild all files: **Project->Rebuild all.**
  d) Load project image: **Project->Debug.**
  e) Run program: **Debug->Go(F5).**
- MDK-ARM
  a) Under the example folder, open \MDK-ARM sub-folder.
  b) Launch the Project.uvproj workspace[(a)].
  c) Rebuild all files: **Project->Rebuild all target files.**
  d) Load project image: **Debug->Start/Stop Debug Session.**
  e) Run program: **Debug->Run (F5)**.
- TrueSTUDIO
  a) Open the TrueSTUDIO toolchain.
  b) Click **File->Switch Workspace->Other** and browse to TrueSTUDIO workspace directory.+
  c) Click **File->Import**, select **General->'Existing Projects into Workspace'** and then click "**Next**".
  d) Browse to the TrueSTUDIO workspace directory, select the project.
  e) Rebuild all project files: select the project in the "**Project explorer**" window then click **Project->build project** menu.
  f) Run program: **Run->Debug (F11).**

a. The workspace name may changes from one example to another.

4.2 Developing your own application

This section describes the steps required to create your own application using STM32CubeF0:

1. **Create your project**

   To create a new project, you can either start from the Template project provided for each board under \Projects\<STM32xxx_yyy>\Templates or from any project available under \Projects\<STM32xxy_yyy>\Examples or \Projects\<STM32xx_yyy>\Applications (where <STM32xxx_yyy> refers to the board name, as example STM32091C_EVAL).

   The Template project provides an empty main loop function. It is a good starting point to get familiar with project settings for STM32CubeF0. The template has the following characteristics:

   – It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components required to develop a code on a given board.

   – It contains the include paths for all the firmware components.

   – It defines the STM32F0 device supported, thus allowing to configure the CMSIS and HAL drivers accordingly.

   – It provides ready-to-use user files pre-configured as shown below:

   HAL is initialized with default timebase with ARM Core SysTick.

   SysTick ISR is implemented for HAL_Delay() purpose.

   System clock configured with the minimum frequency of the device (HSI) for an optimum power consumption.

Note: When copying an existing project to another location, make sure to update the include paths.

2. **Add the necessary middleware components to your project (optional)**

   The available middleware stacks are USB Device library, StemWin, Touch sensing library, FreeRTOS and FatFS. To know which source files you need to add in the project files list, refer to the documentation provided for each middleware. You can also look at the applications Projects\STM32xxx_yyy\Applications\<MW_Stack> (where <MW_Stack> refers to the Middleware stack, as an example USB_Device) to know which sources files and which include paths need to be added.

3. **Configure the firmware components**

   The HAL and middleware components offer a set of build time configuration options using #define macros declared in a header file. A template configuration file is provided within each component, it has to be copied to the project folder (usually the configuration file is named xxx_conf_template.h, the word “_template” need to be removed when copying it to the project folder). The configuration file provides enough information to know the impact of each configuration option; more detailed information is available in the documentation provided for each component.

4. **Start the HAL Library**

   After jumping to the main program, the application code calls *HAL_Init()* API to initialize the HAL Library, which performs the following tasks:

   a) Configuration of the Flash prefetch and SysTick interrupt priority (configured by the user through macros defined in stm32f0xx_hal_conf.h).

   b) Configuration of the SysTick to generate an interrupt each 1 ms at the SysTick TICK_INT_PRIO interrupt priority defined in stm32f0xx_hal_conf.h, which is

clocked by the HSI (at this stage, the clock is not yet configured and thus the system runs from the internal HSI at 8 MHz)

c) Call of HAL_MspInit() callback function defined in stm32f0xx_hal_msp.c user file to performs global low level hardware initializations.

5. Configure the system clock

The system clock configuration is done by calling the two APIs described below:

a) HAL_RCC_OscConfig(): this API configures the internal and/or external oscillators, as well as PLL source and factors. The user can configure one oscillator or all oscillators. The PLL configuration can be skipped if there is no need to run the system at high frequency.

b) HAL_RCC_ClockConfig(): this API configures the system clock source, the Flash memory latency and AHB and APB prescalers.

6. Peripheral initialization

a) Start by writing the peripheral HAL_PPP_MspInit function. Please proceed as follows:

– Enable the peripheral clock.

– Configure the peripheral GPIOs.

– Configure DMA channel and enable DMA interrupt (if needed).

– Enable peripheral interrupt (if needed).

b) Edit stm32xxx_it.c file to call the required interrupt handlers (peripheral and DMA), if needed.

c) Write process complete callback functions if you plan to use peripheral interrupt or DMA.

d) In your main.c file, initialize the peripheral handle structure then call the HAL_PPP_Init() function to initialize your peripheral.

7. Developing your application process

At this stage, your system is ready and you can start developing your application code.

The HAL provides intuitive and ready-to-use APIs to configure the peripheral. It supports polling, interrupts and DMA programming model, to accommodate any application requirements. For more details on how to use each peripheral, refer to the rich set of examples provided.

If your application has real time constraints, you can find a large set of examples showing how to use FreeRTOS and integrate it with all middleware stacks provided within STM32CubeF0. This can be a good starting point to develop your application. You can also refer to STM32F0xx HAL drivers user manual.

Caution: In the default HAL implementation, SysTick timer is time base source. It is used to generate interrupts at regular time intervals. If HAL_Delay() is called from peripheral ISR process, make sure that the SysTick interrupt has higher priority (numerically lower) than the peripheral interrupt. Otherwise, the caller ISR process is blocked. Functions affecting timebase configurations are declared as __weak to make override possible in case of other implementations in user file (using a general purpose timer for example or other time source). For more details please refer to HAL_TimeBase example.

4.3 Using STM32CubeMX to generate the initialization C code

An alternative to steps 1 to 6 described in *Section 4.2* consists in using the STM32CubeMX tool to generate code to initialize system, peripherals and middleware (steps 1 to 6 above) through a step-by-step process:

1. Select the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
2. Configure each required embedded software thanks to a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and the utility performing MCU peripheral configuration (for example GPIO, USART) and middleware stacks (for example USB).
3. Generate the initialization C code based on the configuration selected. This code is ready to use within several development environments. The user code is kept at the next code generation.

For more information, please refer to STM32CubeMX user manual (UM1718).

4.4 Getting STM32CubeF0 release updates

The STM32CubeF0 firmware package comes with an updater utility, STM32CubeUpdater, also available as a menu within STM32CubeMX code generation tool.

The updater solution detects new firmware releases and patches available from st.com and proposes to download them to the user's computer.

4.4.1 Installing and running the STM32CubeUpdater program

Follow the sequence below to install and run the STM32CubeUpdater:

1. To launch the installation, double-click the *SetupSTM32CubeUpdater.exe* file available under \Utilities\PC_Software.
2. Accept the license terms and follow the different installation steps.
3. Upon successful installation, STM32CubeUpdater becomes available as an STMicroelectronics program under Program Files and is automatically launched. The STM32CubeUpdater icon appears in the system tray. Right-click the updater icon and select **Updater Settings** to configure the Updater connection and perform manual or automatic checks. For more details on Updater configuration, refer to section 3 of STM32CubeMX User manual - UM1718).

5 FAQ

5.1 What is the license scheme for the STM32CubeF0 firmware?

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license.

The middleware stacks made by STMicroelectronics (USB Host and Device Libraries, StemWin) come with a licensing model allowing easy reuse, provided it runs on an STMicroelectronics device.

The middleware based on well-known open-source solutions (FreeRTOS and FatFs) have user-friendly license terms. For more details, refer to the license agreement of each middleware.

5.2 What boards are supported by the STM32CubeF0 firmware package?

The STM32CubeF0 firmware package provides BSP drivers and ready-to-use examples for the following STM32F0 boards: STM32072B-EVAL, STM32091C-EVAL, STM32F072B-DISCOVERY, STM32F0308-DISCOVERY, NUCLEO-F070RB, NUCLEO-F072RB, NUCLEO-F091RC, NUCLEO-F030R8, NUCLEO-F031K6 and NUCLEO-F042K6.

5.3 Is there any link with Standard Peripheral Libraries?

The STM32Cube HAL layer is the replacement of the Standard Peripheral Library.

The HAL APIs offer a higher abstraction level compared to the standard peripheral APIs. HAL focuses on peripheral common functionalities rather than hardware. Its higher abstraction level allows to define a set of user friendly APIs that can be easily ported from one product to another.

Although existing Standard Peripheral Libraries are supported, they are not recommended for new designs.

5.4 Does the HAL layer take benefit from interrupts or DMA? How can this be controlled?

Yes. The HAL layer supports three API programming models: polling, interrupt and DMA (with or without interrupt generation).

5.5 Are any examples provided with the ready-to-use toolset projects?

Yes. STM32CubeF0 provides a rich set of examples and applications (more than 50 for STM32091C-EVAL). They come with pre-configured projects for several toolsets: IAR, Keil and GCC.

5.6 How are the product/peripheral specific features managed?

The HAL layer offers extended APIs, that is specific functions as add-ons to the common API to support features available on some products/lines only.

5.7 How can STM32CubeMX generate code based on embedded software?

STM32CubeMX has a built-in knowledge of STM32 microcontrollers, including their peripherals and software. This enables the tool to provide a graphical representation to the user and generate *.h/*.c files based on user configuration.

5.8 How can I get regular updates on the latest STM32CubeF0 firmware releases?

The STM32CubeF0 firmware package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new firmware package updates (new releases or/and patches).

STM32CubeUpdater is also integrated within the STM32CubeMX tool. When using this tool for STM32F0 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeF0 firmware package updates.

For more details, refer to *Section 4.4*.

5.9 How do I set the HAL drivers in Debug mode to debug my application?

To configure the HAL drivers in Debug mode, set the USE_FULL_ASSERT compilation switch in your application project environment. This allows to systematically check the parameters passed to the HAL APIs. The assert_failed() function is called when an error is detected.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
17-Jun-2014	1	Initial release.
30-Sep-2014	2	Added StemWin in *Section 1: STM32CubeF0 main features* and *Section 2: STM32CubeF0 architecture overview*. Updated *Figure 1: STM32CubeF0 firmware components* and *Figure 2: STM32CubeF0 firmware architecture*. Added STM32F091xx and STM32F098xx in *Table 1: Macros for STM32F0 series*. Updated *Figure 3: STM32CubeF0 firmware package structure* and *Figure 4: STM32CubeF0 examples overview*. Added STM32091C-EVAL and NUCLEO-F091RC in *Table 2: Boards for STM32F0 series* and *Table 3: Number of examples available for each board*. Changed STM32F072RB-NUCLEO and STM32F030R8-NUCLEO into NUCLEO-F072RB and STM32F072RB-F030R8 in *Table 3: Number of examples available for each board*. Replaced STM32F071RB by STM32F091RC in *Section 4.1: Running your first example*. Added STemWin in *Section 4.2: Developing your own application* and replaced STM32072B_EVAL by STM32091C_EVAL. Updated *Section 5.1: What is the license scheme for the STM32CubeF0 firmware?* and *Section 5.2: What boards are supported by the STM32CubeF0 firmware package?*, and added *Section 5.9: How do I set the HAL drivers in Debug mode to debug my application?*.
05-Dec-2014	3	Added STM32F030xC, STM32F070x6 and STM32F070xB in *Table 1: Macros for STM32F0 series*. Added NUCLEO-F070RB in *Table 2: Boards for STM32F0 series*, *Table 3: Number of examples available for each board* and *Section 5.2: What boards are supported by the STM32CubeF0 firmware package?*.
10-Sep-2015	4	Updated *Figure 1: STM32CubeF0 firmware components*. Added NUCLEO-F031K6 and NUCLEO-F042K6 board in *Section 3.1: Supported STM32F0 devices and hardware*, *Table 3: Number of examples available for each board* and *Section 5.2: What boards are supported by the STM32CubeF0 firmware package?*.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved